ViSP: A software environment for eye-in-hand visual servoing

E. Marchand

IEEE International Conference on Robotics and Automation, ICRA'99
Vol 4, pp. 3224–3230, Detroit, Michigan, May 1999

ViSP: A Software Environment for Eye-in-Hand Visual Servoing

Éric Marchand
IRISA - INRIA Rennes
Campus de Beaulieu, F-35042 Rennes Cedex
Email: Eric.Marchand@irisa.fr

Abstract

*In this paper, we describe a modular software that allows fast development of eye-in-hand image-based visual servoing applications (*ViSP *states for "Visual Servoing Platform"). Visual servoing consists in specifying a task as the regulation in the image of a set of visual features. Various issues have thus to considered in the design of such application: among these issues we find the control of camera motions and the tracking of visual features. Our environment features a wide class of control skills as well as a library of real-time tracking processes. Some applications that used this modular architecture on a six dof cartesian robot are finally presented.*

1 Overview

Visual servoing is a very important research area in vision-based robotics (see [4] for an extensive review). Despite all the research in this field, it seems that there are no software environment that allows fast prototyping of visual servoing tasks. The main reason is certainly that it usually requires specific hardware (the robot and, most of the time, dedicated image processing boards). The consequence is that the resulting applications are not portable and can be merely adapted to other environment. Today's software design allows to propose elementary components that can be combined to build portable high-level applications. Furthermore, the increasing speed of micro-processors allows the development of real-time image processing algorithms on a simple workstation.

Chaumette, Rives and Espiau [3] proposed to constitute a "*library of canonical vision-based tasks*" for eye-in-hand visual servoing that contents the most classical linkages that are used in practice. Toyama and Hager [9] describe what such a system should be in the case of stereo visual servoing. The proposed system is specified using the same philosophy as the XVision system [7]. This system (called Servomatic) would have been independent from the robot and the tracking algorithms. Following these ways, ViSP, the software environment we present in this paper features all these capabilities: independence with respect to the hardware, simplicity, extendibility, portability. Moreover, ViSP features a large library of elementary positioning tasks wrt. various basic control features (points, lines, circles, spheres, cylinders, etc.) that can be combined together, and an image processing library that allows the tracking of visual cues (dot, segment, ellipse, spline, etc.). This modular platform has been developed in C++ on Unix Workstations.

The remainder of this paper presents the background dealing with the eye-in-hand visual servoing and the tracking algorithms. Software design is then presented in the light of a simple example. Experimental results carried out on a six dof cartesian robots are finally presented.

2 Image Based Visual Servoing

The *image-based visual servoing* consists in specifying a task as the regulation in the image of a set of visual features [6][8]. Embedding visual servoing in the task function approach allows us to take advantage of general results helpful for the analysis and the synthesis of efficient closed loop control schemes. These control issues are now well known, we just give a rapid overview of the visual servoing process.

2.1 Control issues

Let us denote $\mathbf{P}$ the current value of the set of selected visual features[1] used in the visual servoing task and measured from the image at each iteration of the control law. To ensure the convergence of $\mathbf{P}$ to its desired value $\mathbf{P_d}$, we need to know the interaction matrix (also called image Jacobian) $\mathbf{L}_\mathbf{P}^T$ defined by the classical equation [6]:

$$\dot{\mathbf{P}} = \mathbf{L}_\mathbf{P}^T(\mathbf{P}, \mathbf{p})\mathbf{T_c} \tag{1}$$

where $\dot{\mathbf{P}}$ is the time variation of $\mathbf{P}$ due to the camera motion $\mathbf{T_c}$. The parameters $\mathbf{p}$ involved in $\mathbf{L}_\mathbf{P}^T(\mathbf{P}, \mathbf{p})$ represent the depth information between the considered objects and the camera frame.

A vision-based task $\mathbf{e}$ is defined by:

$$\mathbf{e} = \mathbf{W}^+\mathbf{C}(\mathbf{P} - \mathbf{P_d}) + (\mathbf{I} - \mathbf{W}^+\mathbf{W})\mathbf{g}_s^T \tag{2}$$

[1] In the reminder of this paper, we will call these features *"control features"* by opposition to the features tracked in the image sequence (*the "visual cues"*) from which the control features are extracted.

where $\mathbf{C}$, called combination matrix, has to be chosen such that $\mathbf{C}\mathbf{L}_{\mathbf{P}}^{T}(\mathbf{P},\mathbf{p})$ is full rank about the desired trajectory $q_r(t)$. It can be defined as $\mathbf{C} = \mathbf{W}\mathbf{L}_{\mathbf{P}}^{T^+}(\mathbf{P},\mathbf{p})$ ($\mathbf{L}^+$ denotes the pseudo inverse of $\mathbf{L}$). In that case, we set $\mathbf{W}$ as a full rank matrix such that Ker $\mathbf{W}$ = Ker $\mathbf{L}_{\mathbf{P}}^{T}$. If the vision-based task does not constrain all the n robot degrees of freedom, a secondary task $\mathbf{g}_s$ can also be performed. $\mathbf{g}_s$ is the gradient of a cost function h_s to be minimized ($\mathbf{g}_s = \frac{\partial h_s}{\partial \mathbf{r}}$). This cost function is minimized under the constraint that $\mathbf{P} = \mathbf{P_d}$. The two projection operators $\mathbf{W}^+$ and $\mathbf{I} - \mathbf{W}^+\mathbf{W}$ guarantee that the camera motion due to the secondary task is compatible with the regulation of $\mathbf{P}$ to $\mathbf{P_d}$.

In order to make $\mathbf{e}$ exponentially decrease and then behave like a first order decoupled system, we get:

$$\mathbf{T_c} = -\lambda\mathbf{e} - \mathbf{W}^+\widehat{\frac{\partial \mathbf{e_1}}{\partial t}} - (\mathbf{I} - \mathbf{W}^+\mathbf{W})\frac{\partial \mathbf{g}_s^T}{\partial t} \qquad (3)$$

where:

- $\mathbf{T_c}$ is the camera velocity;
- λ is the proportional coefficient involved in the exponential convergence of $\mathbf{e}$;
- $\widehat{\frac{\partial \mathbf{e_1}}{\partial t}}$ (with $\mathbf{e_1} = \mathbf{C}(\mathbf{P} - \mathbf{P_d})$) represents an estimation of a possible autonomous target motion.

2.2 A library of visual servoing skills

One of the difficulty in visual servoing is to derived the interaction matrix $\mathbf{L}^T$ corresponding to the selected control features. A systematic method has been proposed to derived analytically the interaction matrix of a set of control features defined upon geometrical primitives [6][3]. Any kind of visual information can be considered within the same visual servoing task (coordinates of points, line orientation, surface (or more generally inertial moments), distance, etc.)

Knowing these interaction matrices, the construction of elementary visual servoing tasks is straightforward. As explained in [3] a large library of elementary skills can be proposed. The current version of ViSP allows to define X-to-X feature-based tasks with X = {point, line, sphere, cylinder, circle, etc.}. Using these elementary positioning skills more complex tasks can be considered by stacking the elementary matrices. For example if we want to build a positioning task wrt. to a segment, defined by two points $\mathbf{P_1}$ and $\mathbf{P_2}$, the resulting interaction matrix will be defined by:

$$\mathbf{L}_{\mathbf{P}}^{T} = \begin{bmatrix} \mathbf{L}_{\mathbf{P_1}}^{T} \\ \mathbf{L}_{\mathbf{P_2}}^{T} \end{bmatrix}$$

where $\mathbf{L}_{\mathbf{P_i}}^{T}$ is defined, if $\mathbf{P_i} = (X, Y)$ and z is its depth, by:

$$\mathbf{L}_{\mathbf{P_i}}^{T} = \begin{pmatrix} -1/z & 0 & X/z & XY & -(1+X^2) & Y \\ 0 & -1/z & Y/z & 1+Y^2 & -XY & -X \end{pmatrix}$$

This way, more feature-based tasks can be simply added to the library.

An other important feature is the capability to introduce a secondary task. This secondary task can be a solution to the impossibility to plan the camera trajectory by introducing some constraints in its motion. We proposed a large library of secondary tasks from trajectory tracking to occlusions avoidance and joint limits/singularities avoidance.

Tracking capabilities have also been integrated. When the target is moving, an estimation of the autonomous target motion is required in order to avoid tracking errors.

3 Tracking Visual Cues

Parallel to the development of the platform with respect to control part of visual servoing, we have to develop algorithms dealing with tracking issues. The visual servoing formalism allows to use more complex features or combination of features. ViSP allows to consider, besides the usual "dots", visual cues such as lines, ellipses, or more complex curves. Information available in the description of these shapes are then used to describe the control features.

We have developed fast real-time tracking processes. Few systems features real-time capabilities on a simple workstation. The XVision system [7] is a good example of such systems, however, it does not features all the tracking capabilities we wanted[2]. In our case, we decided to use the ME (moving edges) algorithm [2] adapted to the tracking of parametric curves. It is a local approach that allows to match moving contours. Previous works have been done to use this algorithm to track line segments [1] on a dedicated IP board.

3.1 General Algorithm

One of the advantages of the method is that it does not require any edge extraction, furthermore it can be implemented with convolution efficiency and can therefore ensure a real-time computation [2]. As we want an algorithm that is fast, reliable, robust to partial occlusions and to false matches, we decided to track only a list L^t of pixels along the considered edge and then to determine, by a robust least square approach, the equation of the support primitive that fits these data.

We will not described the initialization process in details, let us just say that we are able to initialize the pixels list L^t. Then, for each pixel we estimate the direction of the tangent to the edge $\widehat{\theta}$. The process consists in searching for the correspondent P_i^{t+1} in image I^{t+1} of each pixel $P_i^t \in L^t$. We determine a 1D search area $Q_i^j, j \in [-J, J]$ in the direction of the normal to the contour δ. For each pixel P_i^t of the list L^t, and for each position Q_i^j lying in the direction δ we compute the matching criterion corresponding to a log-likelihood ratio ζ^j. This is nothing but the absolute sum of the convolution values computed at P_i and Q_i^j using a pre-determined mask M_θ function of the orientation of the contour. New position P_i^{t+1} is given by:

$$Q_i^{j^*} = \arg\max_{j \in [-J,J]} \zeta^j \text{ with } \zeta^j = \mid I(P_i) * M_\theta + I(Q_i^j) * M_\theta \mid$$

[2] Even if, in many way, it is far more complete that the system we propose.

providing that the value ζ^{j^*} is greater than a threshold λ. Then pixel P_i^{t+1} given by $Q_i^{j^*}$ is stored in L^{t+1}. A new list of pixels is obtained.

Finally, given the list L^{t+1} the new parameters of the feature are computed using a least squares technique.

3.2 Tracking Visual cues

Line segments. The simplest case we consider is the line segment [1]. The representation considered for the line are the polar coordinates (ρ, θ):

$$x \cos\theta + y \sin\theta - \rho = 0$$

This case is very simple as the direction θ is directly given by the parameters of the features. The choice of the convolution mask is then straightforward. A points insertion processes either in the middle of the segment, to deal with partial occlusions or miss-tracking, and at the extremities of the segment to deal with sliding movements has been introduced in the tracking method.

Ellipses. Dealing with the ellipse many representation can be defined, we chose to use the coefficients K_i that are obtained from the polynomial equation of an ellipse:

$$K_0 x^2 + K_1 y^2 + 2K_2 xy + 2K_3 x + 2K_4 y + K_5 = 0$$

The ellipse correspond to the case $K_2^2 < K_0 K_1$. The parameters K_i can be estimated from the pixels of the list L_t using a least square method. From the parameters K_i we can derived other representations as the parameterization $(X_c, Y_x, \mu_{11}, \mu_{02}, \mu_{20})$ based on the normalized inertial moments.

Splines. A spline is defined by a parametric equation:

$$Q(t) = \sum_{j=-d}^{n-1} \alpha_j B_j(t), \quad t \in [0,1]$$

where the α_j are the control knots of the spline, d is the degree of the spline ($d = 3$ for a cubic spline) and B_j spline basis function. Since the number p of tracked point is usually greater than the number of desired control knots n, a least square method is used.

Discussion. The proposed tracking approach algorithms based on the ME algorithm allows a real-time tracking of geometric features in an image sequence. It is robust with respect to partial occlusions and shadows. However, as a local algorithm, its robustness cannot be ensure in complex scenes with highly textured environment.

4 Software environment

As already stated, while developing this software, our goal was to allow a portable (independent from the hardware), fast and reliable prototyping of visual servoing applications. The first part of this section presents the internal architecture of the system and how it has been implemented. Describing the full implementation of the software is out of reach in this paper, therefore, we will focus on the notion of extendibility and portability. The second part describes how to use the available libraries from a end-user point of view. Let us note that all the functionalities described in this section have been implemented and are *fully operational*.

4.1 An overview of the ViSP architecture

To fulfill the extendibility and portability requirements we divided the platform into three different parts (the tracking parts, a library of control features, and the controller itself) and we widely use C++ capabilities (templates, derivation, inheritance, virtual classes, etc.).

As can be seen on the network of Figure 1, each library of the ViSP environment is indeed extensible. In the controller library, the `CServoAfma` class is derived from a `CServo` virtual class and is specific to our six dof Afma robot. It redefines some pure virtual methods defined in `CServo` such as robot motion orders (*i.e.*, the `CServoAfma::UpdateCameraVelocity` specific to a given robot) and inherits all the methods and attributes of `CServo` (*i.e.*, generic control issues). Adding a new robot is then straightforward. In the same way some specific frame grabber classes (here `CSunvideo` or `CEdixia`, that are our frame grabbers) can be derived from a generic `CFrameGrabber` class. This two examples also show that the platform is independent from the hardware and then portable.

The extendibility can be mainly seen in the control features library. Each specific control feature is derived from a virtual class `CBaseFeatures`. This class mainly defines a few variables (*e.g.*, a vector that describes the parameters $\mathbf{P}$ and $\mathbf{p}$) and a set of virtual members functions that are features dependent (*e.g.*, the way to compute the image Jacobian $\mathbf{L}$ or the virtual function that allows the tracking of the feature in the images). It is important to note that all the relations between the controller library and the features library is done through this class. The virtual functions define in `CBaseFeatures` can be directly used by the controlled even if they are not yet defined. The consequence is that the controller library never know the nature of the manipulated features. Another consequence is therefore that it is absolutely not necessary to modify the controller library when adding a new feature. On the other hand, when adding a new feature in the control features library, the programmer must define the number of visual information, the way to compute the image Jacobian, etc. This is done at a lower level (*e.g.*, `CPoint`, `CLine`, ...). Some member functions of these derived classes remain virtual. Indeed a control feature can be defined by many visual cues, member functions like `Track(...)` cannot be defined at this level. Therefore, a few classes derive from the feature classes in order to supply the users with many tracking capabilities. For example a point can be define as a simple dot (`CPointDots` in Figure 1) or as the intersection of two lines (`CPointTwoLines`). In any case, using the inheritance mechanism the controller knows which function it has to use. In conclusion, adding new capabilities within the control features library can therefore be done

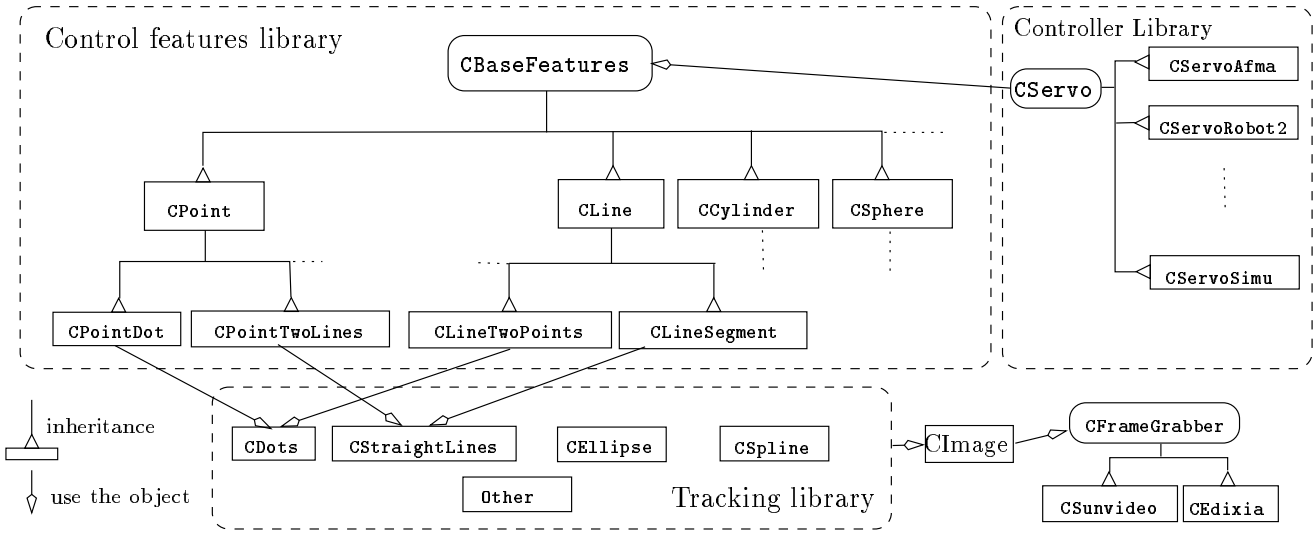

Figure 1: ViSP structure of classes

at two levels: adding a new control feature (*e.g.*, a circle in Figure 1) or adding a new way to link this feature to the visual cues (for example, defines a point as the center of an ellipse).

4.2 ViSP from a end-user point of view

Our other claim was that ViSP is simple to use. We will therefore describe the software environment, from the end-user point of view, in the light of two simple examples implemented using ViSP. The first example is a positioning task wrt. a line.

```
 0 main() {
 1  CImage I ;
 2  I.InitAcqImage(Sunvideo()) ;
 3  I.AcqImage() ;
 4
 5  CLineSegment line ;         // or CLineTwoDots line ;
 7  line.InitTracking(I) ;
 8
 9  CLine Ld(0,PI/2) ;  // Init here the desired visual
10                      // features Ld (centered/horizontal)
 1
 2  CServoAfma task ;
 3  task.AddLink(line,Ld) ;
 4  task.InitInteractionMatrix() ;
 5
 6  while(...) {
 7    CColVector Tc(6) ;
 8
 9    I.AcqImage() ;
20    task.GetCurrent(I) ;
 1    Tc = -0.2 * task.TaskFunction() ;  // Tc = -l L^+(s-sd)
 2    task.UpdateCameraVelocity(CAMERA,Tc) ;
 3  }
 4 }
```

Line 2 defines the image acquisition protocol (here through the sunvideo frame grabber). The `CImage::AcqImage()` function puts the bitmap in the computer memory where the tracking process will be done. A control feature, a line, is defined (5-6) as a segment (`CLineSegment`). This means that the visual cue is a line segment. Line 9 defines the desired position of the visual feature in the image. The controlled `CServoAfma` derived from the `CServo` virtual class is defined line 12. Line 13 defines a link between the current position (`line`) of the visual feature in the image and the desired position (`Ld`) while line 14 creates the corresponding interaction matrix (here a 2×6 matrix). Note that more complex tasks can be defined by "stacking" other links using the `CServo::AddLink` method. It is straightforward to write the loop itself. It features the image acquisition (19) and the current visual features extraction or tracking (20). Finally the vision based task e is computed using the `CServo::TaskFunction()` method and the new robot velocity orders $\mathbf{T_c} = -\lambda \mathbf{e}$ (22).

The task may be more complex. Let us consider, a curves following task (see the experimental results in the next section). This problem can be divided in two subtasks. The primary task consists in servoing on the tangent to the curve (*i.e.*, maintain this tangent horizontal and centered in the image). The positioning skill used in this experiment is therefore a line-to-line link. Indeed, image features used here are $\mathbf{P} = (\rho, \theta)$ where ρ and θ are the parameters of a line that is nothing but the tangent to the curve (2 dof are then controlled). The secondary task is a trajectory tracking at a given velocity in the X direction and is defined by a cost function $h_s = X - X_0 - V_x t$. Image processing consists here in tracking a spline in the image sequence and to compute the equation of the tangent to the curve from which we can control the camera motion. From a control point of view, this task is similar to the previous one. However, in our software environment there are no direct relations between the tangent to a curve (that is, here, the visual cue) and a line (that is the control feature). As explained in the previous example, ViSP usually allows to avoid an explicit access to the

trackers, but the number of relations visual cues/control features is virtually infinite. Therefore a direct access to the trackers is sometimes necessary. The spline tracker, here, is defined in line 1 whereas the visual feature (a straight line) is associated to this tracker in line 4. Then in the control loop, the spline is tracked in each frame (the `CSpline::Track(CImage &I)` method in line 10) and the new control feature is computed (line 11) and introduced in the controller (the `CServo::NewCurrent(...)` method in line 12). This visual servoing task also features the use of a secondary task. Vector $\mathbf{g}$ is the gradient of h_s, $\mathbf{g} = V_x$ and is combined with the primary task using the projection operation $\mathbf{I} - \mathbf{W}^+\mathbf{W}$ (line 14). These simple examples allow us to show the importance of the three libraries: the trackers library (define by variable in line 1), the control features library (line 3), and the controller library (line 6) and how they interact with each other.

```
main() {
   CImage I ;
 1 CSpline S(CUBICSPLINE) ;
 2 S.InitTracking(I) ;

 3 CLine L ;
 4 L = S.Tangent(0,0) ;
 5 CLine Ld(0,PI/2) ;

 6 CServo task ;
 7 task.AddLink(L,Ld) ;
 8 task.InitInteractionMatrix() ;

   while(...) {
     CColVector Tc(6), g(6)
 9   I.AcqImage() ;
10   S.Track(I) ;          // Track the spline (visual cue)
11   L = S.Tangent(0,0) ; // compute the tangent

12   task.NewCurrent(L) ; // define the new control feature
13   g[0] = Vx ;          // secondary task
14   Tc = -0.2* (task.TaskFunction() + task.I_WpW*g);
15   task.UpdateCameraLocation(CAMERA,Tc) ;
   }
}
```

5 Some Applications

Table 1 sums up the different elementary positioning tasks that have been implemented using VISP. Most of these results are now classic and will not be described here. We just propose results for less classic experiments, *i.e.* positioning wrt. to a sphere and curves following. Other applications implemented with VISP are structure from controlled motion (point, sphere), joint limits and singularities avoidance, occlusions avoidance, ...

The application presented in this section has been implemented at IRISA on a six dof cartesian robot. Image processing (described in Section 3) and control law (Section 2) are performed on a Sun Ultra Sparc 1 (170 Mhz). The frame grabber was a Sunvideo board. Tests performed on the Sun show that a line can be tracked in 3.6 ms (with 40 pixels in the list L^t and a maximum displacement $J = \pm 10$) whereas an ellipse is tracked in 6.5 ms. A spline with 40 pixels in L^t and 15 control knots is tracked in less than 10 ms.

Curves following task. The principle of this experiment has been described in the previous paragraph. Let us just add that the goal was to follow a long pipe (2 meters long) that features three 90^o corners. Figure 3 show three images acquired during this task in the first corner (note the tracked spline and the tangent in the image). Figure 2a depicts the translational velocities. The bottom curves is the due to the secondary task. In order to avoid high rotational velocity in high curvature area, the velocity V_x has been defined as $V_x = V_{max} \exp(-\alpha(\theta - \theta^*)^2)$. This explain the noise in this plot and the three "$\bigcap$" that can be observed (due to the three corners). Figures 2b and c depict the errors $\theta - \theta_d$ and $\rho - \rho_d$ observed for the selected visual features. The important error in θ while crossing the high curvature area are due to the fact that the curvature is not predicted. This problem is very similar to the tracking errors that can be observed in a target tracking task. Figure 2d depicts the XY camera trajectory that reflects the shape of the pipe.

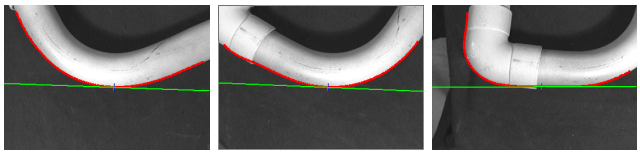

Figure 3: Pipe following task: 3 images acquired during the task and tracked curve (red) and tangent to the curve (green)

Let us note that a similar problem has already been addressed in [5]. But even if the tracking was done using snakes, the visual features used in the control were points.

Positioning wrt. a sphere. In this second experiment, we want to servo on a sphere (a ping-pong ball). The task is to observe the sphere as a centered circle in the image with a given radius and then to move the camera at a constant distance from the sphere center. The projection of a sphere in the image plane is an ellipse. Image processing consists in tracking this ellipse (here on a non-uniform environment). We chose as visual features the inertial moments of this ellipse (*i.e.*, $X_c, Y_x, \mu_{11}, \mu_{02}, \mu_{20}$). Since the real radius of the ping-pong ball is known (19 mm), the secondary motion (along the $\vec{X}$ axis) does not introduce any perturbations in the primary task Each iteration is performed in 40 ms.

6 Conclusion and future work

VISP is a *fully functional* modular architecture that allows fast development of visual servoing applications. It is mainly composed of three C-callable libraries: two dedicated to control issues (one of control processes and one of canonical vision-based tasks that contains the most classical linkage [3]) and one dedicated to real-time tracking (based on the Moving edges algorithm). This

objects	*Control features* (see [6] for details)	*Visual Cues* dots	lines	ellipse	B-Spline
point	point $\mathbf{P} = (X, Y)$	×	×		
line	line $\mathbf{P} = (\rho, \theta)$		×		× (tangent)
circle	ellipse $\mathbf{P} = (X_c, Y_c, \mu_{11}, \mu_{20}, \mu_{02})$			×	
sphere	ellipse $\mathbf{P} = (X_c, Y_c, \mu_{11}, \mu_{20}, \mu_{02})$			×	
cylinder	lines $\mathbf{P} = (\rho_1, \theta_1, \rho_2, \theta_2)$		×		
square	points $\mathbf{P}_i = (X_i, Y_i), i = 1..4$	×	×		
	lines $\mathbf{P}_i = (\rho_i, \theta_i), i = 1..4$		×		
quadrilateral	moments/orientations $\mathbf{P} = (X_c, Y_c, \mu_{00}, \alpha_1, \alpha_2, \alpha_3)$	×			

Table 1: Elementary positioning tasks using ViSP: tracked cues and control features.

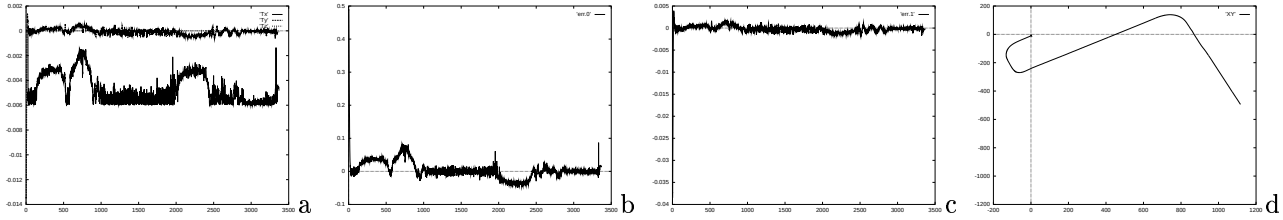

Figure 2: Pipe following task: (a) Translational camera velocity (b) Error $\theta - \theta_d$ and (c) $\rho - \rho_d$ (d) XY camera trajectory

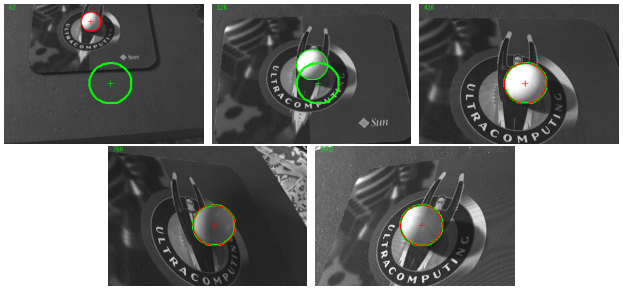

Figure 4: Tracking ellipse and servo on a sphere: $t = 62$, $t = 126$, $t = 416$, $t = 760$, $t = 1602$

software has already been used for the development of a large number of applications. Let us finally note that ViSP also features a virtual six dof robot that allows to simulate visual servoing experiments. Dealing with vision-based control, many new features can be added to this software. Only eye-in-hand 2D visual servoing with respect to geometrical features have been proposed. Therefore other visual servoing technics can be introduced such as stereo visual servoing, or visual servoing wrt. dynamic informations or motion, etc. Furthermore, now that a set of basic tools are available, it is necessary to be able to deal with high-level mission that combines many tasks.

References

[1] S. Boukir, P. Bouthemy, F. Chaumette, and D. Juvin. A local method for contour matching and its parallel implementation. *Machine Vision and Application*, 10(5/6):321–330, April 1998.

[2] P. Bouthemy. A maximum likelihood framework for determining moving edges. *IEEE Trans. on PAMI*, 11(5):499–511, May 1989.

[3] F. Chaumette, P. Rives, and B. Espiau. Classification and realization of the different vision-based tasks. *in* K. Hashimoto, editor, *Visual Servoing*, pp. 199–228, World Scientific, Singapor, 1993.

[4] P.I. Corke. Visual control of robot manipulator. K. Hashimoto, editor, *Visual Servoing*, pp. 1–32, World Scientific, Singapor, 1994.

[5] E. Coste-Manière, P. Couvignou, and P. Khosla. Visual servoing in the task-function framework: a contour following task *Journal of Intelligent and Robotics Systems*, 12:1–21, July 1995.

[6] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Trans. on Robotics and Automation*, 8(3):313–326, June 1992.

[7] G. Hager and K. Toyama. The XVision system: A general-purpose substrate for portable real-time vision applications. *Computer Vision and Image Understanding*, 69(1):23–37, January 1998.

[8] K. Hashimoto, editor. *Visual Servoing: Real Time Control of Robot Manipulators Based on Visual Sensory Feedback*. World Scientific Series in Robotics and Automated Systems, Vol 7, World Scientific Press, Singapor, 1993.

[9] K. Toyama, G. Hager, and J. Wang. Servomatic: A modular system for robust positioning using stereo visual servoing. In *Proc. of the International Conference on Robotics and Automation*, pp. 2636–2643, Minneapolis, April 1996.